University at Buffalo
Department of Computer Science and and Engineering
CSE 573 - Computer Vision and Image Processing

Fall 2023

Home Work #4
Due Date: 12/05/23, 11:59PM

1 Face Detection in the Wild (40 Points)

For this, you have to utilize any face detection algorithm available in opencv library (version >= 4.5.4.) and perform face detection.

Given a face detection dataset composed of hundreds of images, the goal is to detect faces contained in the images. The detector should be able to locate the faces in any testing image. Figure 1 shows an example of performing face detection. We will use a subset (will be provided) of FDDB [**?**] as the dataset for this homework. You can use any face detection modules available in OpenCV.

Figure 1: An example of performing face detection. The detected faces are annotated using gray bounding boxes.

1.0.1 Libraries permitted and prohibited

- Any API provided by OpenCV.

  You may NOT use any internet examples that directly focuses on face detection using the OpenCV APIs.

- You may NOT use any APIs that can perform face detection or machine learning from other libraries outside of OpenCV

1.0.2 Data and Evaluation

You will be given zip file that contains 100 images along with ground-truth annotations (validation folder). You can evaluate each of your models performances on this validation set or use it to further improve your detection. During testing, you need to report results on another 100 images (test folder) without ground truth annotations. Please read the README.md files provided in the homework folder and refer to the script in it for running and validating your code.

Your implementation should be in the function `detect_faces()`, in the file `task1.py`. The function should detect faces in the given image and return the bounding boxes of the detected faces in a list (maybe more than 1 face in a images). The bounding box should be in the format of `[x, y, width, height]`, `x` and `y` are the top-left corner of the bounding box; `width` and `height` are the width and height of the bounding box, respectively. `x`, `y`, `width` and `height` should be float numbers. Consider origin (0,0) to be the top-left corner of the image and `x` increases to the right and `y` increases to the bottom. See the code for the rest of input and output formats.

1.0.3 Evaluation Rubric

Rubric: 40 points

For F1 score computed using ComputeFBeta.py. (40 points for the F1 score.)
Yourscore = min(1, 1.25∗F1) ∗ 40.

1.1 Instructions (Please read this very carefully!):

- Please implement all your code in given file **"task1.py"**. Please do NOT make any changes to any file except **"task1.py"**.
- You can only use the given libraries provided in the **"task1.py"** code. You can not make any new imports.
- Identical code will be treated as plagiarism. Please work it out independently.
- For code raising "RuntimeError", the grade will be ZERO.

2 Image Panorama (60 points)

This task aims to stitch multiple images into one panoramic photo. As shown in Fig. 2, the given images might be non-overlapping or multiply-overlapped (overlapping two or more other images). The shape of the output image after stitching might be irregular since you are not allowed to crop either of the transformed images. There are no restrictions regarding the method you use to stitch photos into a panoramic photo. For this project, you can assume the following:

- Your code will need to be able to stitch together four or more images and you will not know that in advance.
- You can assume that IF an image is to be part of the panorama, it will overlap at least one other image and by at least 20%.
- Images that do not overlap with any other image can be ignored.
- Images can overlap with multiple images.
- Although the Figure below shows horizontal panoramas, your five images can be stitched together in any way.
- You are only expected to produce one overall image.
- While some of the most modern techniques may use a spherical projection for better panoramas, you are free to assume that basic 2D planer transformations are sufficient for this project.

Steps:

- Extract features for each image, match features, and determine the spatial overlaps of the images automatically. For instance, among four images, if image1 overlaps with image3, image4, then your overlap array should be $\begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$. Follow the instructions in the provided script and save the overlap result as a $N * N$ one-hot array in the JSON file. (30 points)
- Conduct image transformation and stitch all into one panoramic photo. Save the photo as the instructions specified in the script. (30 points)

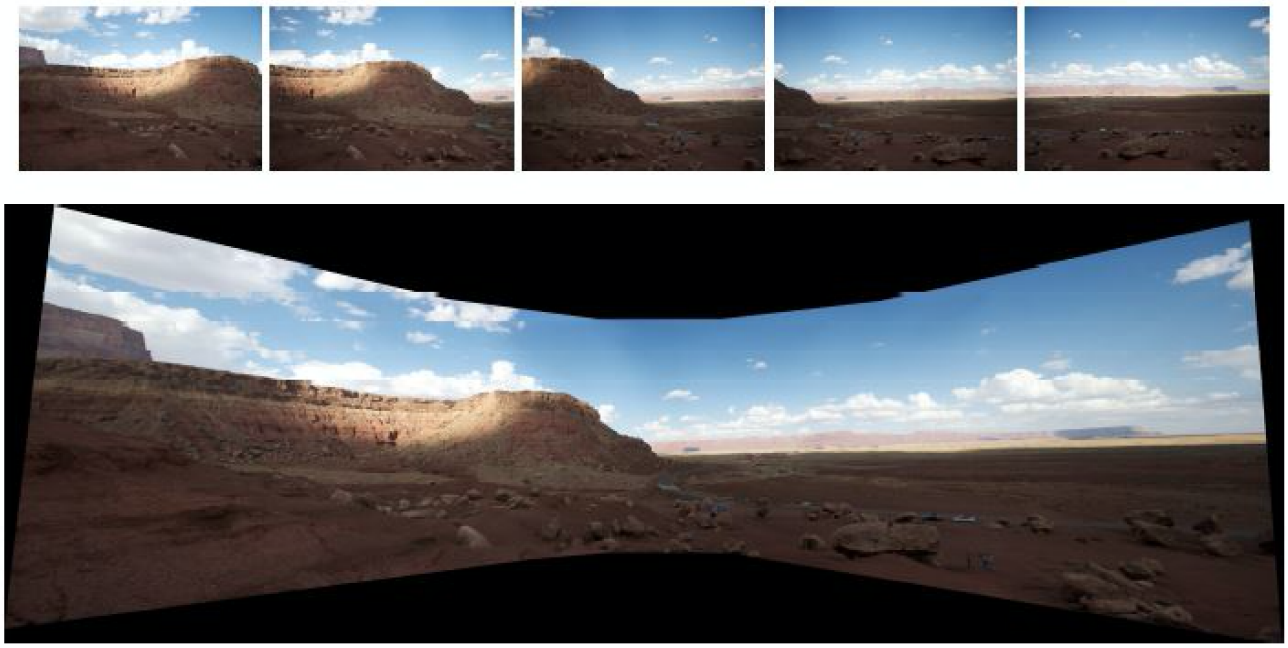

Figure 2: Example of image panoramas.

2.1 Instructions (Please read this very carefully!):

- Do not modify the code provided.
- You can use the basic Python and OpenCV libraries for feature extraction and homography computation (cv2.findHomography()), but you may NOT use any libraries related to establishing candidate matches (APIs that have "match" in their names) or stitching (APIs that have "stitch" or "mosaic" in their names).You may make new imports based on these guidelines.
- You should map and blend pixels across images based on the homography computation to obtain the final mosaiced images for all parts of this homework.
- You may NOT copy code from other projects on the internet.
- For code raising "RuntimeError", the grade will be ZERO.
- If you decide to use SIFT for feature extraction, please note that it has been patented and it has been removed from OpenCV3, but it is included in OpenCV2.
- All work should be your own. You are not permitted to copy or plagiarize code from other students or from online discussion boards, websites, or archives. Any attempt to do so will result in failure.

3 Submission Instructions

- Unlimited number of submissions is allowed and only the latest submission will be used for grading.
- To submit your code and result, Please run **"pack_submission.sh"** to pack your code and result into a zip file. You can find the command line in **"README.md"**. Note that when packing your submission, the script would run your code before packing. The resulting zip file is only file you need to submit.
- The packed submission file should be named **"submission_YourUBITName.zip"**, and it should contain 6 files, named **"task1.py"**, **"task2.py"**, and **"results_task1.json"**, **"results_task1_val.json"**, **"task2_overlap.txt"** and **"task2_result.png"**. If not, there is something wrong with your code/filename, please go back and check.